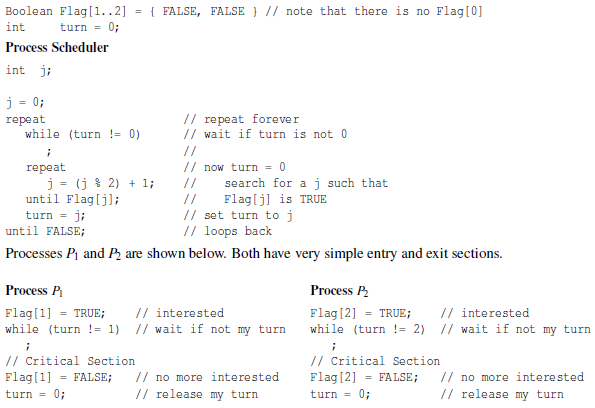
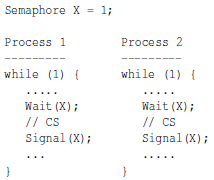
Exam 2 Fall 2015

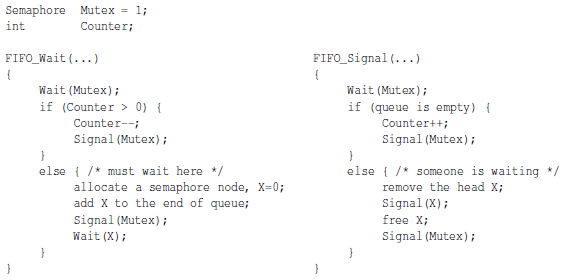
1. The following is a solution to the critical section problem. It has two shared variables flag[] and turn and a process Scheduler started before the processes. The Scheduler waits until turn becomes 0, then the repeat-until loop searches for a j such that flag[j] is true. Finally, turn is set to the value of j and loops back. Prove that this solution satisfies mutual exclusion and bounded waiting. State the bound, then prove the condition.



1. Define the meaning of a race condition. Use execution sequences.
2. Consider the following implementation of mutual exclusion with a semaphore. Prove the implementation satisfies mutual exclusion.



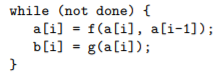
1. A programmer designed a FIFO semaphore so that the waiting processes can be released in a first-in-first-out order. This FIFO semaphore has an integer counter Counter, a queue of semaphores, and procedures FIFO\_Wait() and FIFO\_Signal(). A semaphore Mutes with initial value 1 is also used. FIFO\_Wait() uses Mutex to lock the procedure and checks Counter. If Counter is positive, FIFO\_Wait() decreases Counter by one, unlocks the procedure, and returns. If Counter is zero, a semaphore X with initial value 0 is allocated and added to the end of the queue of semaphores. Then FIFO\_Wait() releases the procedure and lets the caller wait on X. procedure FIFO\_Signal() first locks the procedure and checks if the semaphore queue is empty. If so, FIFO\_Signal() increases Counter by one, unlocks the procedure, and returns. If there is a waiting process in the queue the head of the queue is removed and signaled so that the only waiting process on that semaphore continue. Then, this semaphore node is freed and the procedure is unlocked. Finally, the initialization procedure FIFO\_Init() not shown below sets the counter to an initial value and the queue to empty. Discuss the correctness of this solution. If you think it correctly implements a FIFO semaphore, provide a convincing argument. Otherwise, discuss why it is wrong with an execution sequence.



1. A simplified SPOOL system has three processes: Spool-in, Spool-out, and Process. They share a spool device, say a disk. Spool-in reads in input from a slow input device and copies it to the spool device, Spool-out sends the print output from the spool device to a slow output device, and Process is a user program that reads in its input from and writes its output to the spool device. To be more efficient, the spool device is divided into a number of slots and each read and write operation reads and writes exactly one slot. Once a slot is read (by Process) or printed (by Spool-out) the space occupied by this slot is considered free and can be re-used. The following are rules for performing a spooling operation:
   1. Initially, the spool device is empty
   2. As long as the spool device has an empty slot, Spool-in will read the input and copy it to the spool device. If all slots are used, Spool-in blocks until there are free slots
   3. Process reads its input from the spool device if there are slots that have been filled with input data by Spool-in; otherwise, Process blocks until new input data becomes available. After reading an input, Process will generate some output, one slot at a time. Process also blocks until there are empty slots for output.
   4. As long as the spool device has output slots, Spool-out will read and send them to the output device. Spool-out blocks until output data becomes available
   5. Reading from and writing into a slot is guaranteed to be mutually exclusive

Under what conditions will this system have a deadlock? Provide an execution sequence and elaborate.

1. A multithreaded program has two global arrays, a and b, and a number of threads that execute concurrently. Thread Ti runs the following code. Declare semaphores with initial values and add wait and signal calls to thread Ti so that it will compute the results correctly.



1. Design a class Group in C++, a constructor, and a method Group\_wait() that fulfill the following:
   1. The constructor Group(int n) takes a positive integer argument n, and initializes a private variable in class Group to have the value of n. The value of n will not change during execution.
   2. Method Group\_wait(void) takes no arguments. A thread that calls Group\_wait() blocks if the number of threads being blocked is less than n-1. Then the nth calling thread releases all n-1 blocked threads and all n threads continue. Note that the system has more than n threads. For example, suppose n is initialized to 3. The first two threads that call Group\_wait() block. When the third thread calls Group\_wait(), the two blocked threads are released, and all three threads continue.

Use semaphores only to implement class Group and method Group\_wait().